

# The Evolution of Evolutionary Software: Intelligent Rhythm Generation in Kinetic Engine

Arne Eigenfeldt

School for the Contemporary Arts, Simon Fraser University  
Burnaby, Canada  
[arne\\_e@sfu.ca](mailto:arne_e@sfu.ca)

**Abstract.** This paper presents an evolutionary music software system that generates complex rhythmic polyphony in performance. A population of rhythms is derived from analysis of source material, using a first order Markov chain derived from subdivision transitions. The population evolves in performance, and each generation is analysed to provide rules for subsequent generations.

**Keywords.** rhythm generation, genetic algorithms, recombination, realtime systems

## 1 Introduction

Kinetic Engine [1, 2, 3] is an evolving interactive musical performance software system that generates complex rhythmic polyphony using musically intelligent algorithms. In its three incarnations to date, it has been used as an installation, a networked performance using ten computers, as an instrument within an ensemble of improvising musicians, an intelligent controller for a 12-armed musical percussion robot, and as a composer of a score for percussion quartet. Addressing some of the limitations in earlier versions, Kinetic Engine version 3 (KE3) uses an evolving population of rhythms produced using a genetic algorithm, which are derived from analysis of source material.

Genetic algorithms (GA) have been used in a variety of ways to create music, both in performance (realtime) and in the studio: KE3 will be discussed in relation to some of these applications. Biles, in his discussion of GenJam [4], points out many of the idiosyncratic factors of using GAs in a musical context, including the notion that the fitness test of any generated data derives from aesthetic judgment, which is inherently difficult to encode [5]. As Martins and Miranda point out, artists may use such techniques not as “tools to generate efficient solutions to problems automatically”, but instead, “composers need tools to explore a vast space of possible outcomes” [6]. They also point out that “generating music from algorithms that were not designed for music seldom produces significant pieces of music”, and instead, composers should modify A-Life algorithms to address musical issues [6]. KE3 is one such system, in which a modified GA is employed for musical means: the goal is not any single solution, but the continual evolution of a given data set that is heard as it evolves.

Rather than beginning from a random population, KE3 begins with an initial population derived through analysis of source material. A separate program module is run to extract relevant musical information from user provided MIDI files. This material can be very explicit (i.e. a set of patterns to use for a section of music) or general (i.e. a transcription of an entire piece of music). The Analysis module parses the material for various aspects (described later), which are passed to multiagent Player modules as XML files. The Player agents, upon initialisation, generate initial populations that closely resemble the rhythmic aspects of the source material. Individuals within the population are chosen by various means (described later) for performance. Evolution occurs in isolation within each agent, and in realtime, during performance. Agents keep track of those individuals that are chosen for performance; heard individuals become more likely to be culled. Since all population-members are deemed acceptable - thus avoiding any direct fitness test - the system does not choose individuals that are allowed to remain alive, but instead chooses individuals to cull.

After each generation has been bred, the new population is analysed, and this new analysis becomes the rule-set for generating new members of the population; as such, the rule-set itself is evolutionary.

Although diversity depends upon the source material (the basis of the original population), it is further maintained through two mutation methods: the substitution of null rhythms (rests) at the beginning or end

of the individual, and the substitution of related chromosomes (rhythmic subdivisions) within the individual. The amount of mutation is under performance control.

Section 2 presents background information, including a brief description of the context of interactive computer music, the difference between realtime composition and improvisation, and earlier versions of Kinetic Engine; Section 3 describes previous research in the field; Section 4 provides a detailed description of the latest version of Kinetic Engine and its relationship to previous systems; Section 5 suggests future directions; Section 6 presents some conclusions.

## **2 Background**

### **2.1 Interactive computer music**

Realtime computer music, in which the computer makes compositional decisions in performance that react to composer/performer interactions, has tended to fall within the domain of improvisatory systems, and has been pursued by those composer/performers interested in more mercurial aspects of music creation<sup>1</sup>. In an effort to model the “unpredictability” of improvisation, constrained random procedures have been incorporated so that the musical surface can be both varied, yet easily controlled [7]; for example, choosing pitches randomly from a user determined scale. Higher level changes, at the gestural level, require more deterministic programming, or greater user control; for example, changing scales, chords, rhythmic units, etc. Formal cohesion, or large-scale structural logic offered by recapitulation and restatement, remains with the composer/performer, through a type of compositional intervention.

### **2.2 Realtime composition versus improvisation**

It is no coincidence that a great deal of realtime computer music has fallen into the same amorphous formal trap as much improvised music, due, in part, to the restrictions on human memory: it is simply too difficult for musicians to retain the detailed musical information created earlier in performance. Al Biles, in describing his own improvisational playing with GenJam, states “I find that a delay of four measures can be too long because it is hard to remember what I played four bars ago” [8].

With the use of computers, the ability to compose during performance becomes conceivable. Realtime composition [9] is distinct from improvisation in a number of ways, but one chief distinction is musical memory<sup>2</sup>. KE3 separates these two creative methods by incorporating predefined polyphonic musical structures as source data, and the ability to control and shape their development (composition), as well as allowing spontaneous control during performance, which does not affect future actions (improvisation). In practical terms, composed elements generate the populations, while improvised elements can influence how the populations are explored.

### **2.3 Previous systems**

Kinetic Engine began as an effort to place more high-level compositional responsibility within software. In attempting to incorporate aspects of artificial intelligence into music performance software, the decision was made to limit the exploration to rhythmic development and interaction.

#### **2.3.1 Kinetic Engine v.1**

As mentioned, realtime systems tend to use constrained random methods to generate, or vary, musical material. However, the notion of “groove” - which one can consider as the interaction of corporeal rhythms that afford a sense of cyclical movement, and makes listeners want to tap their feet - is not something that can be generated randomly. Brown recognises this when he states that rhythms generated through the use of

---

<sup>1</sup> See [17] for an overview of early interactive systems and methodologies.

<sup>2</sup> See [9] for a detailed discussion of the differences between realtime composition and improvisation.

cellular automata “often result in a lack of pulse or metre. While this might be intellectually fascinating, it is only occasionally successful from the perspective of a common aesthetic” [10].

Kinetic Engine version 1 (KE1) focused on the interrelationship of simple parts to create continuously varying rhythmic interplay between four virtual players. Basic rules were defined as to how players generated individual parts; more complex rules were defined as to how these parts interacted.

KE1 was successful as an installation, in that it required no human performer supervision or control; however, it could not be used as a performance instrument.

### 2.3.2 Kinetic Engine v.2

Kinetic Engine version 2 allowed for performative control, and furthered the notion of intelligent interaction between individual players through autonomous multiagents. Agents generated specific rhythms in response to a changing environment. Once these rhythms had been generated, agents “listened” to one another, and altered their patterns based upon these relationships. Since the rule-set for rhythm generation remained static, the resulting music, while initially engaging, remained essentially homogeneous. Furthermore, since the complex interactions were based upon seeded probabilities, the music was essentially improvisatory and afforded limited compositional control.

## 2.4 Kinetic Engine v.3

The latest version of Kinetic Engine attempts to balance spontaneous change with the ability to alter and adapt its rule-set through predetermined (composed) decisions. As such, KE3 offers the potential for both realtime composition through recombination [11] as well as improvisation through generative means. Section 4 describes the software in detail.

## 3. Previous Research

One of the first realtime music applications of genetic algorithms, *GenJam* [4] performed jazz improvisations by creating a population of melodic phrases which had been evolved in one of two ways: under the interactive guidance of a human mentor, or by eliminating the fitness bottleneck entirely by populating the database with pre-selected data. Biles states that *GenJam*’s intention was to be received artistically, rather than conceptually: “Instead of challenging an *audience* artistically, *GenJam* tried to challenge an audience technically by achieving an accessible and convincing performance from a computer” [8]. KE2 is similar in this regard, in that it is a realtime tool created by a composer for use in performance, with musical, rather than purely conceptual, goals.

More recently, Martins and Miranda [12] created an A-Life system in which a collective rhythmic database evolved among a group of musical agents. They state at the outset that the system was based upon language imitation games; as such, it was more experimental and conceptual, rather than a system that produced successful music. The initial database of rhythms was randomly generated, and individuals were passed between agents in order to evolve a repertoire.

Alfonseca et al. [13] used a genetic algorithm to generate music in a pre-defined style. GAs have proven to be useful in these situations, since the goal (stylistic emulation) is objective, and thus a fitness function can be more clearly defined. Similarly, Mararis et al. [14] created artificial music critics, trained via a neural net using 992 musical works, that evaluated the artifacts created by an evolutionary music composer: the goal of the evolutionary composer was to create “aesthetically pleasing” variations on Bach’s *Invention #13*. Significant correlations were found between the artificial critics and 23 human subjects. As with Alfonseca, it is easier to test a system’s success given such objective criteria: personal systems, whose goal is to create *new* music, are more difficult to judge, since their success depends upon the ability to recreate the composer’s aesthetic, one which may not be judged as “aesthetically pleasing” by listeners. This, according to Nic Collins, is “playing the composer card” (personal communication).

More relevant to the research presented here is the work of Weinberg et al. [15], who used a GA to create a population of melodies used by a robotic performer in response to a human improviser. A number of melodic excerpts of variable lengths and styles were transcribed from an improvising pianist, and served

as an initial population. Individuals were chosen in realtime in response to input melodies. The fitness test measured similarity to the input phrase.

Waschka's *GenDash* [16] employs evolutionary computation algorithms to "help compose" musical works. Waschka, a composer, has used the software to create a large number of concert works, and whose statement that "algorithmic techniques should be interesting, useful and serve the composer – not the other way around" is a sentiment echoed by this author. Similarities to KE3 include an initial population chosen by the user, and a stochastic fitness function. *GenDash* is not a realtime system.

## 4 Kinetic Engine version 3

KE3 is unique in several ways. First, although its population is comprised of rhythmic representations, its evolution occurs via transitions found within the population, rather than altering the population directly. Second, transition data found in individuals - used to generate successive generations - is not accumulated into a single table, but remains discrete. This avoids a type of neutralizing, or averaging, of the data, and results in closer variations to individual rhythmic phrases. Third, unlike many machine learning systems, KE3 does not attempt to learn general rules from a large dataset; instead, the author recognises that successful music is entirely context-dependent. As such, specific examples are given to the system in order to deduce rules specific to a desired section of music: these rules are immediately forgotten.

The use of GAs within composition, even in realtime, is well established; even the avoidance of a using a fitness function has been implemented previously. KE3's initial design did not include a GA: it's inclusion literally evolved during development from artistic needs. Separating analysis from performance allowed for the creation of a potential pool of available material for Player agents - the musical development of this material naturally suggested GA and its unique features.

Furthermore, the use of a GA permitted the concept of musical memory to enter the system: rhythms do not emerge and disappear (as they do with most improvisatory systems, including KE2); instead, they evolve over time, while potentially retaining older versions. The ability to resurrect older individuals by re-introducing them into the population is a further example of exploiting this concept.

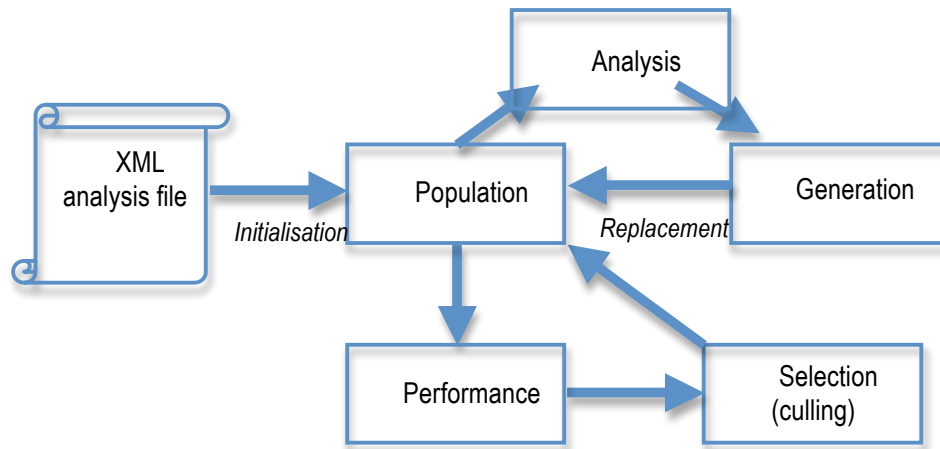
Almost all of the design decisions have an artistic motivation, including the avoidance of the fitness test. Although a fitness test is not directly applied, evaluation functions definitely remain. In KE3, the fitness of any individual will be its suitability to the current musical environment: since the environment is constantly changing, the suitability of any individual will likewise change. Therefore, those individuals that meet the immediate requirements are selected for performance (see section 4.6). Furthermore, fitness will change over time through repetition: musical composition is a balance between repetition and variation, familiarity and novelty. The more an individual is selected for performance (and is heard), the less useful it becomes. However, continual variation through the introduction of new data is also not ideal, since some degree of repetition is required; thus, some older individuals need to survive in order to maintain musical cohesion<sup>3</sup>.

### 4.1 Design

See figure 1 for an outline of the design of KE3.

---

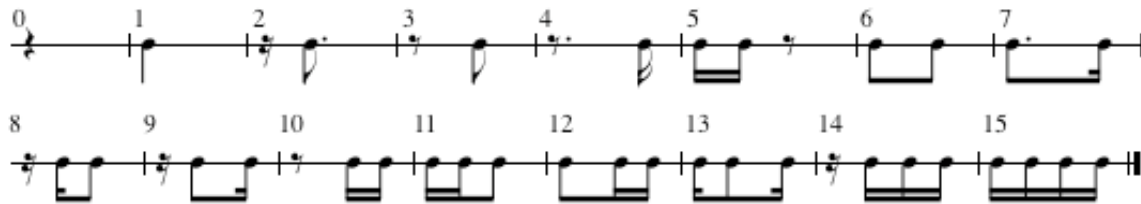
<sup>3</sup> Clearly, these statements are reflections of the author's compositional aesthetic: readers may differ in their own musical values.



**Fig. 1.** The design scheme for Kinetic Engine version 3.

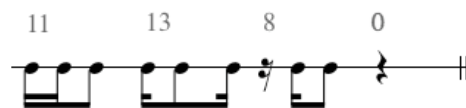
## 4.2 Representation

Kinetic Engine uses a chromosome structure for representing rhythmic phrases of indices into a database of potential rhythmic subdivisions of a beat. See figure 2 for the first sixteen representations.



**Fig. 2.** The first sixteen subdivisions within the subdivision array.

Thus, the following rhythm is represented internally within Kinetic Engine as (11 13 8 0):



**Fig. 3.** Example rhythm and its representation.

Individuals can be of varying length, based upon the phrase lengths detected by the analysis module. Pitches data will not be discussed here.

## 4.3 Initialisation and population analysis



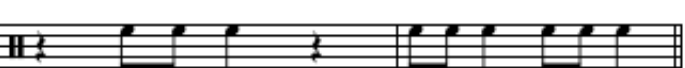

Loading an XML analysis file constitutes initialisation. As well as providing the initial population, data from this file includes a similarity analysis between phrases (used by the Player to determine how *much* variation will occur through selection) and sectional periods (used to determine how *often* to make variations).

Every new population is immediately analysed after generation for rules on how to breed the next population: initialisation produces the initial population, and thus the first analysis. Transitions between subdivision representations are generated using Markov analysis, and stored in a multi-dimensional matrix in order to separate individual beat densities. Population analysis also includes an individual's *density* (the number of events per rhythmic pattern compared to the maximum possible), and *complexity* (the degree of

syncopation within the individual beats and placement within the pattern): both are used by the Player in its selection criteria.

Once this initial analysis is complete, the next population is bred: approximately four children are spawned from each parent, to a maximum of thirty-two individuals. When complete, the Player agent can begin selecting individuals from the population to perform. During performance, a similarity matrix (see Table 1) is generated in the background, using various pattern-matching algorithms and fuzzy logic comparisons. This data is used by the Player in the selection of new individuals at variation points.

**Table 1.** Four example individuals (A1 is a child of A), and their ratings for density, complexity, and similarity to one another. Note that B is considered slightly more complex than others because its onsets are spread out across the pattern.

<p>A</p> 	<p>Density: 0.41 Complexity: 0.23 Similarity: 1.0, 0.86, 0.31, 0.06</p>
<p>A1</p> 	<p>Density: 0.44 Complexity: 0.23 Similarity: 0.86, 1.0, 0.27, 0.07</p>
<p>B</p> 	<p>Density: 0.28 Complexity: 0.26 Similarity: 0.31, 0.27, 1.0, 0.2</p>
<p>C</p> 	<p>Density: 0.25 Complexity: 0.2 Similarity: 0.06, 0.07, 0.2, 1.0</p>

Initialisation may take place several times during a performance: whenever new analysis data is loaded into the system. This will result in the complete loss of an existing population, and the generation of an entirely new one. In musical terms, this occurs at each new musical section, where distinct changes are required in material. While this may appear somewhat erratic from an A-Life standpoint, it addresses a major problem with a great deal of realtime computer music, and improvisation in general: the inability to quickly change musical direction (or “turn on a dime” as one musician described it).

#### 4.4 Selection for elimination

Since all individuals in a population are children of strong solutions, all are considered fit to live (and be heard). Depending upon the musical environment, in most cases only a small percentage of the population will actually be heard. Individuals selected seldom or not at all by the Player remain “fresh”, while those that are selected many times are considered “stale”. The number of times an individual is selected for performance is recorded, and those with higher rankings in this regard have a greater chance of elimination.

The elimination algorithm is a variation of the roulette wheel selection (RWS): the amount of “culling” is user defined (the default is 50%). The population is sorted by usage, and a random value is exponentially scaled to select overused patterns. This method does not guarantee elimination of weak candidates (those that are stale), nor the prolongation of strong candidates (those that remain fresh). Musically, this maintains an unpredictable balance between the repetition of previous material with the introduction of new material.

#### 4.5 Generation and replacement (reproduction)

Breeding is user initiated during performance, although an automated switch is triggered if any one member is heard too often (an arbitrary value equal to the size of the population itself).

Crossover, a standard evolutionary technique in GA in which portions of two individuals are spliced together, is not a usual developmental technique in music. Therefore, instead of directly combining parental information, KE3 uses the genetic material (the probability transitions between rhythms) of a single parent to intelligently create a variation. Special attention is paid to beginnings and endings of rhythmic phrases through independent transition matrices for these points.



Fig. 4. A parent and four children.

The result of restricting evolution to single parents is a population expanding along divergent paths. Since culling is essentially stochastic - those patterns that are selected for performance, and thus rated higher in the repetition score, are based upon immediate requirements (such as density and complexity) that are outside the reproduction process - the potential for genetic drift is high. Furthermore, due to the small population, allele may be entirely lost, thus limiting potential divergence. For this reason, it is possible during breeding to reintroduce ancestors - the original individuals from source data - to the genetic pool. Although the number and exact members selected are stochastically chosen, this essentially restocks the genetic pool, preventing, or at least limiting, both convergence and divergence.

As with all GAs, mutation is necessary to avoid stagnation and allow unpredictable elements to enter the population. KE3 has two basic mutation algorithms. The first technique is a Stravinsky-like substitution of rests for notes at the beginnings and ends of patterns; this ensures that an individual's length remains constant, as well as the specific placement of subdivisions on intended beats. The second algorithm substitutes similar chromosomes (similarity being determined by an algorithm that compares weighted onsets and density) within the patterns themselves. In both cases, the degree of mutation is constrained during performance: this amounts to another compositional control, as it has lasting effects.



Fig. 5. A chromosome (a), and two similar chromosomes, (b and c) and two dissimilar (d and e). (b) and (c) are thus potential mutation substitutions, whereas (d) and (e) are not. Similarity coefficients are displayed in relation to (a).

#### 4.6 Selection for Performance

KE3 uses a sequencer model for time, in which measures and beats are counted using a standard transport mechanism. The Player agent continually generates an array of future events - considered the agent's Intention - derived from the available rhythms and pitches in its population. Which individuals are chosen depends upon the musical environment: player agents look to the user-set global density and complexity parameters, and use a K-nearest neighbor algorithm to choose the individual from the population that best matches these values.

Each agent calculates a variation vector, which determines how often to change individual patterns, and how much to change by. This vector is based upon analysis of the original source material: if the source material is deemed to have a high diversity in terms of density, complexity, and similarity, the agent will assume a similar diversity in choosing individuals from the population to perform. Similarly, the amount of repetition within the source material will determine how often new individuals will be chosen. Variation is, therefore, under compositional control.

#### **4.6.1 Generative control and adaptation**

In certain situations, individuals in the existing population may not adequately meet the musical requirements set by the user during performance. In these cases, temporary adaptations may be made to the data contained in the agent's Intention by adjusting the complexity (syncopation) and density (number of events per beat) to match the user-set parameters. As these variations occur outside the population, they are not retained within it, and can thus be considered improvisational responses to the environment.

## **5. Future Directions**

Several improvements can be made to Kinetic Engine version 3, some of which have resulted in the beginnings of Kinetic Engine version 4. The ability to accept audio input as source material has already begun, taking into account the subtle variations in timing and volume that can be considered expressive performance. These parameters will be analysed and used by the Player agents.

Somewhat ironically, the most logical extension of Kinetic Engine has been on the "to do" list the longest, that of incorporating pitch and harmonic analysis to create non-percussive: this is the basis of Kinetic Engine version 4. The initial limitation in KE1 on rhythm was for the practical reason of constraining the search domain; however, as new areas of rhythmic complexity and structuring have continued to present themselves, the "limit" upon rhythm has not proven "limiting" at all.

## **6. Conclusion**

The ability of genetic algorithms to produce consistently good solutions is one reason that they have become a useful A-Life technique for computer-assisted composition. However, the necessity of testing the quality of population members has reduced its usefulness in realtime situations. Several researchers have come up with alternatives to the fitness bottleneck, and KE3 uses one method first used by Biles in *GenJam*: pre-populating the database with pre-selected data. However, unlike Biles, the data is derived from analysis, rather than the direct incorporation of source material.

The latest version of Kinetic Engine continues with many of the techniques implemented in its earlier incarnations, including the use of autonomous agents and fuzzy logic, in an effort to bring musical intelligence to the realtime decision-making. The use of a population of GA-produced rhythms and interrelated pitches as a potential pool of patterns from which an autonomous Player agent can choose is a major addition to this evolving software. Furthermore, it allows for a potential balance between the instantaneous change of improvisation, and the predetermination of composition.

## **7. Acknowledgements**

This research was undertaken through a Research/Creation grant from Canada's Social Science and Humanities Research Council (SSHRC).

Thanks to Eduardo Miranda, Alexis Kirke, and Torsten Anders for their suggestions and comments during a brief residency at the University of Plymouth's Future Music Lab.

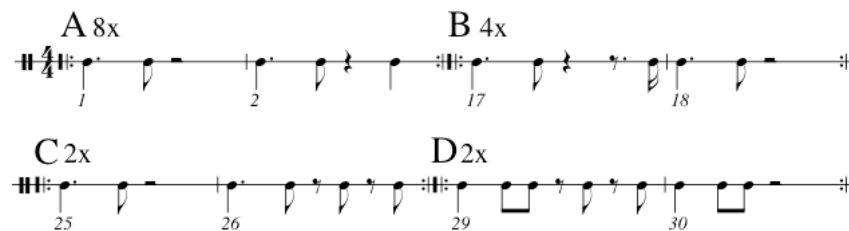


## 8. References

1. Eigenfeldt, A.: Kinetic Engine: Toward an Intelligent Improvising Instrument. In: *Proceedings of the Sound and Music Computing Conference*, Marseille (2006)
2. Eigenfeldt, A.: *Drum Circle*: Intelligent Agents in Max/MSP. In: *Proceedings of the International Computer Music Conference*, Copenhagen (2007)
3. Eigenfeldt, A.: Multiagent Modeling of Complex Rhythmic Interactions in Realtime Performance. In: *Sounds of Artificial Life: Breeding Music with Digital Biology*, A-R Editions (2008)
4. Biles, J. A.: GenJam: A Genetic Algorithm for Generating Jazz Solos. In: *Proceedings of the International Computer Music Conference*, San Francisco (1994)
5. Biles, J. A.: Autonomous GenJam: Eliminating the Fitness Bottleneck by Eliminating Fitness. In: *Proceedings of the Genetic and Evolutionary Computation Conference Workshop Program*, San Francisco (2001)
6. Martins, J., Miranda, E.R.: Emergent rhythmic phrases in an A-Life environment. In: *Proceedings of ECAL Workshop on Music and Artificial Life*, Lisbon (2007)
7. Chadabe, J.: Interactive Composing. In: *Computer Music Journal*, vol. 8. (1984)
8. Biles, J. A.: GenJam: Evolutionary Computation Gets a Gig. In: *Proceedings of the Conference for Information Technology Curriculum*, Rochester (2002)
9. Eigenfeldt, A.: Intelligent Realtime Composition, [http://cec.concordia.ca/econtact/10\\_4/](http://cec.concordia.ca/econtact/10_4/) (2008)
10. Brown, A.: Exploring Rhythmic Automata. In: *Applications On Evolutionary Computing*, vol. 3449, (2005)
11. Cope, D.: Experiments in Musical Intelligence. Middleton, A-R Editions (1996)
12. Martins, J., Miranda, E. R.: A Connectionist Architecture for the Evolution of Rhythms. In: *Proceedings of EvoWorkshops, Lecture Notes in Computer Science*, Berlin, Springer-Verlag, Budapest (2006)
13. Alfonseca, M., Cebrian, M., Ortega, A.: A simple genetic algorithm for music generation by means of algorithmic information theory. in: *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, (2007)
14. Manaris, B., Roos, P., Machado, P., Krehbiel, D., Pellicoro, L., and Romero, J.: A Corpus- based Hybrid Approach to Music Analysis and Composition. In: *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*, Vancouver (2007)
15. Weinberg, G., Godfrey, M., Rae, A., and Rhoads, J.: A Real-Time Genetic Algorithm in Human-Robot Musical Improvisation. In: *Computer Music Modeling and Retrieval, Sense of Sounds*, Springer, Berlin (2008)
16. Waschka, R.: Composing with Genetic Algorithms: GenDash. In: *Evolutionary Computer Music*, Springer, London (2007)
17. Eigenfeldt, A.: Realtime Composition or Computer Improvisation - A Composer's Search for Intelligent Tools in Interactive Computer Music, <http://www.ems-network.org/> (2007)

## 9 Appendices

### Appendix 1: Example source material for one agent



**Fig. 6.** An example of source material, created in Finale, for Kinetic Engine version 3. This gives KE3 a “range of options” from which to compose material, including beat transitions, density, complexity, as well as a range of overall similarity for the variation vector. Certain patterns can be weighted higher by repetition (i.e. pattern A).

## Appendix 2: Example 1<sup>st</sup> Generation

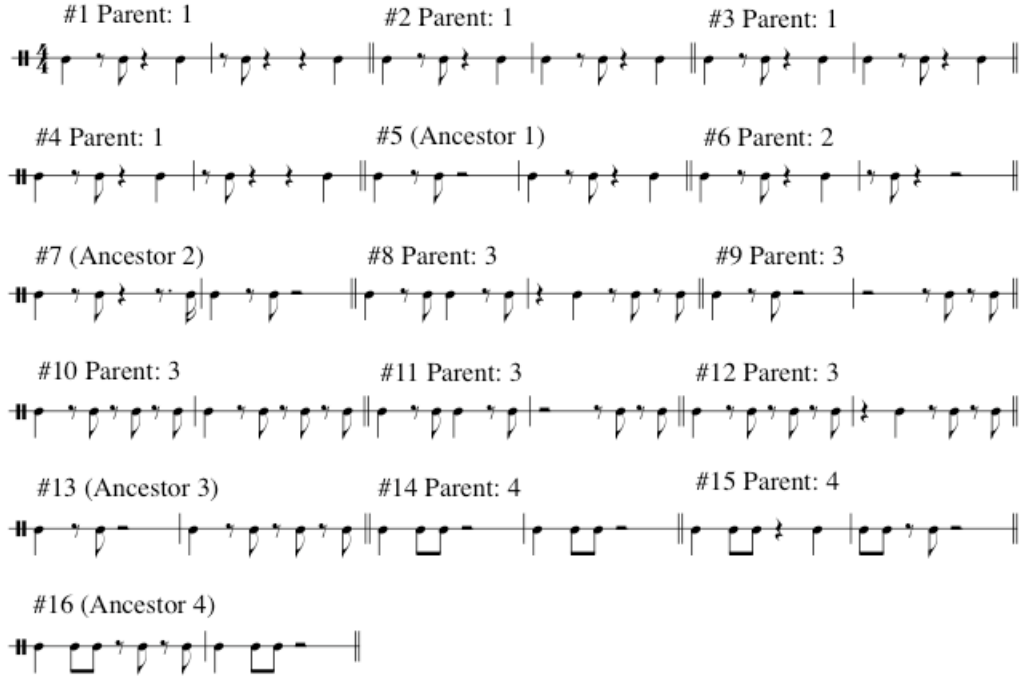


Fig. 7. An example first generation from the source given in appendix 1.

## Appendix 3: Example Intention phrase

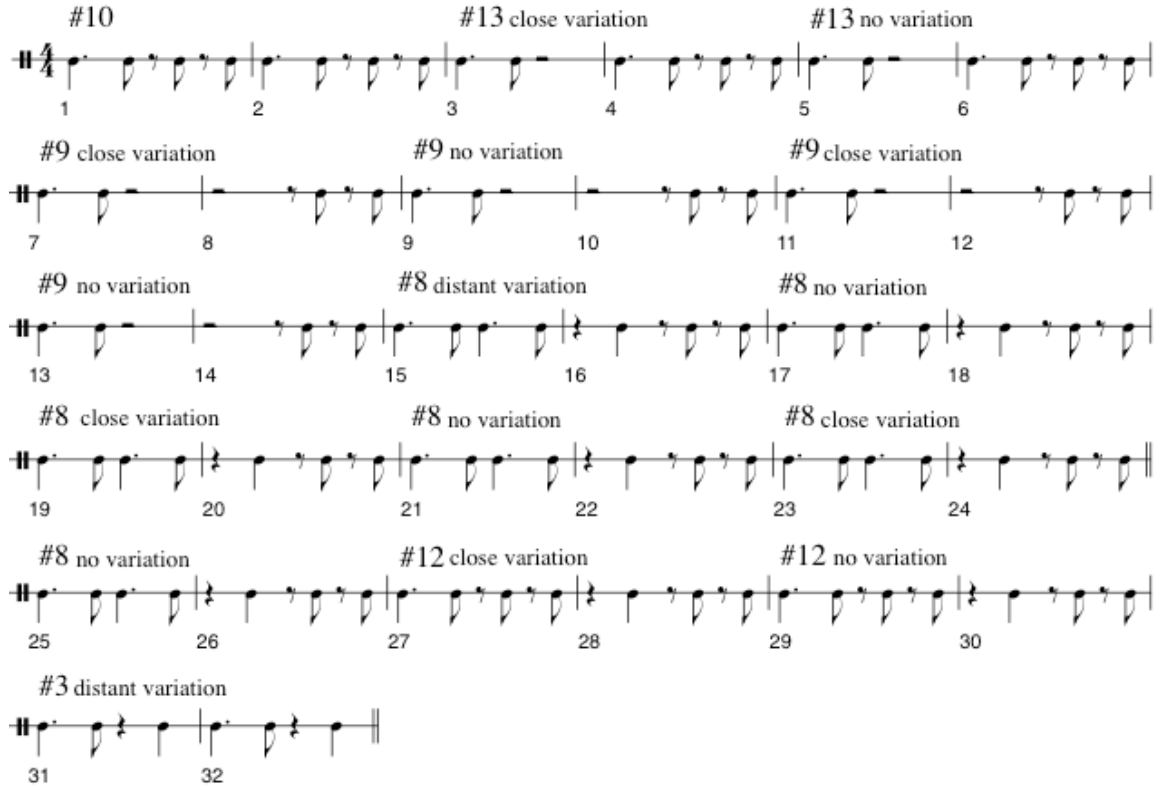


Fig. 8. An example intention generated from the source material in appendix 1. Population individuals are indicated, as are the degree of variation at variation points. Variation vector for phrase was (0 1 0 1 0 1 0 2 0 1 0 1 0 2), where 0 is no variation, 1 is a close variation, and 2 is a distant variation.